



---

## IMPLEMENTATION OF THE EVENT TAG DB

*P. Christakoglou*

*University of Athens - CERN*

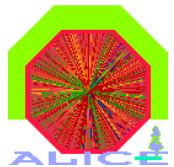


# OUTLINE

---



- Introduction
  - Motivation
  - Event Tag DB @ STAR
- Grid Collector
  - GC's components
  - Architecture
- Bitmap Index
- Alice tag prototype
  - Tag Class description
- Proposed scenarios and architecture
- Summary & next steps



# INTRODUCTION



## MOTIVATION

---



- Users want to analyze only some "interesting events" and not the whole data sample.
- Events are stored in millions of files.
- These files are distributed on many SEs.
- Typical analysis jobs read one event at a time and thus loop over many events that have no significance to the corresponding analysis (mainly time consuming).
- We want to implement a system that will store all the necessary information so that a user can query it in order to **select and analyze the delivered events of interest.**

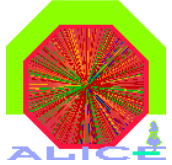


# STAR DATABASE SYSTEM

---



- STAR uses MySQL as its main db engine.
- They've created several dbs in order to monitor different procedures:
  - **CONDITIONS DB:** *Online* record of the measured operating conditions of the detectors.
  - **CONFIGURATIONS DB:** *Online* repository of detector settings for configuring runs.
  - **SCALERS DB:** *Online* record of scaler quantities from trigger & RHIC.
  - **RUNLOG DB:** *Online* record of each experimental run
  - **CALIBRATION DB:** *Offline* record of detector-signal corrections.
  - **GEOMETRY DB:** *Offline* record of geometrical & material constants for the STAR systems
  - **RUNPARAM/CODEPARAM DB:** *Offline* code-constants for reconstruction & analyses.
  - **TAG DB:** *Offline* event level summary record for query-access to data.
  - **PRODUCTION DB :** *Offline* record of production processing and file locations.



## STAR TAG DB

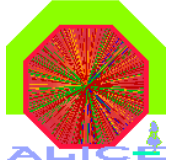
---



- Tag data are stored in root files (tree structures).
- Physics tags are created during reconstruction.
- The corresponding fields are defined and described in C structures.
- They have different tag branches with respect to the physics topic:
  - FLOW TAG
  - E-by-E TAG
  - HEAVY FLAVOR TAG
  - HIGH Pt TAG
  - PERIPHERAL COLLISIONS TAG
  - SPECTRA TAG
  - STRANGENESS TAG
  - HBT TAG
- The physics related tags are around 200.



## GRID COLLECTOR



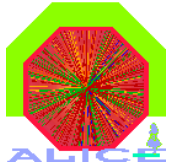
## GRID COLLECTOR

---



- Grid Collector (GC) is a set of software designed to provide file-transparent event access for analysis programs.
- Users specify their requests for events as sets of conditions on physically meaningful attributes (such as trigger, production version, multiplicity etc.)
- GC resolves the given conditions into a list of files containing the events.
- It locates the files and then transfers them if necessary.
- Finally the selected events are passed to the analysis programs.

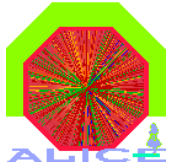




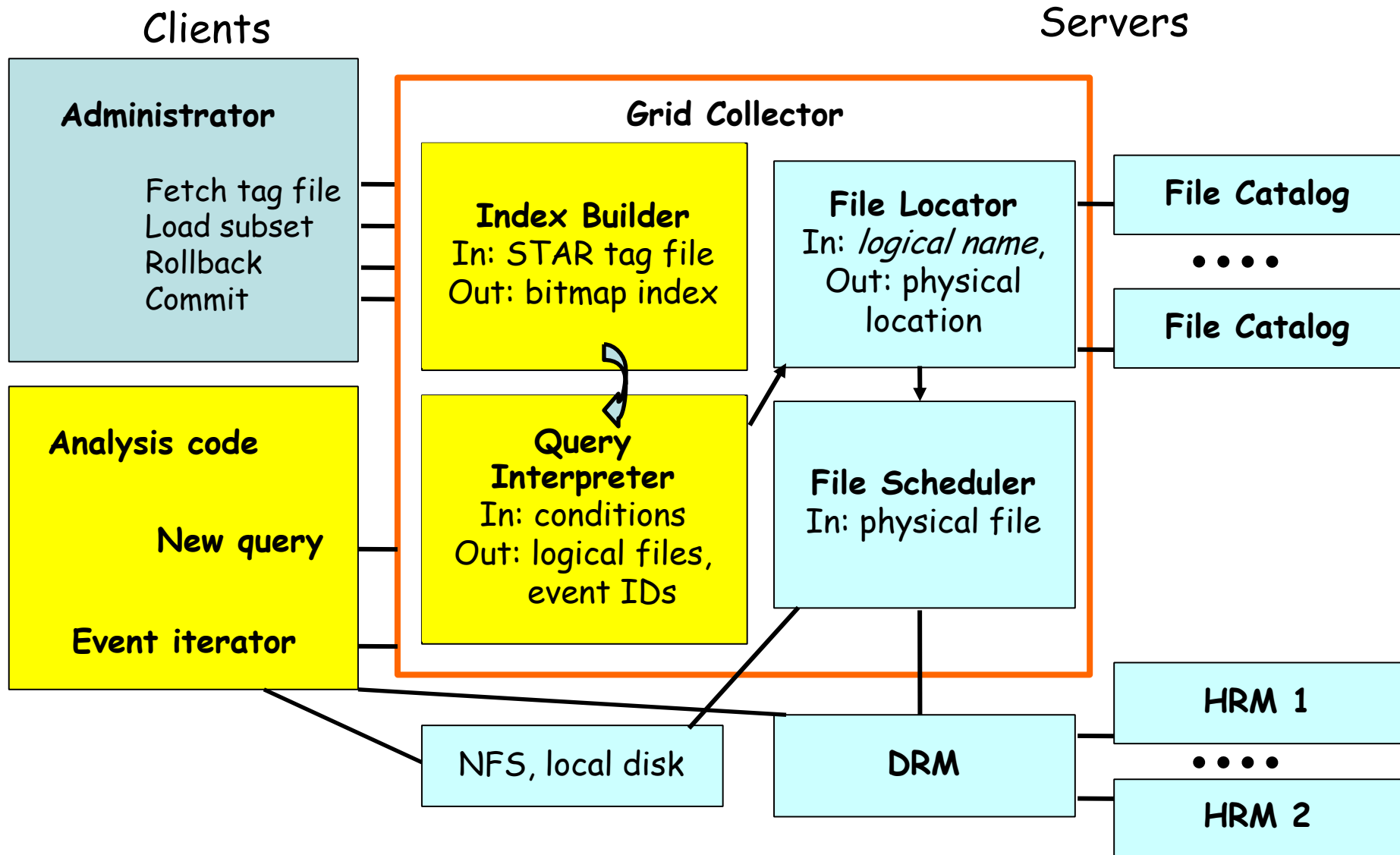
## GC COMPONENTS

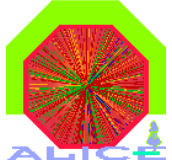


- Build indices for each attribute listed in the tag.root file.
  - Index builder.
- Translate the selection criteria provided by the user into a list of files and events in the files.
  - Query interpreter.
- Locate the files containing the events of interest.
  - Event Catalog, file & replica catalogs.
- Prepare disk space and transfer.
  - Prepare disk space for the files.
    - ➔ Disk Resource Manager (DRM).
  - Transfer the files to the disks.
    - ➔ Hierarchical Resource Manager (HRM) to access HPSS.
    - ➔ On-demand transfers from HRM to DRM.
  - Recover from any errors.
    - ➔ HRM recovers from HPSS failures.
    - ➔ DRM recovers from network transfer failures.
- Read the events of interest from files.
  - Event Iterator with fast forward capability.
- Remove the files.
  - DRM performs garbage collection using pinning and lifetime.



# GRID COLLECTOR ARCHITECTURE





## MAIN GC FEATURES (ALICE)

---



### ● QUERY INTERPRETER

- It takes as an input the tag root files.
- The index builder reads the values of each attribute and generates the corresponding indices.
- These indices are stored in separate files.
- The QI then, translates the selection criteria provided by the user into a list of files and events in the files that satisfy them.

### ● EVENT ITERATOR

- Glues the analysis framework to the GC.
- Its constructor takes a string argument which is a simplified version of the SQL select statement.
- The EI retrieves the files and passes the selected events to the analysis code.



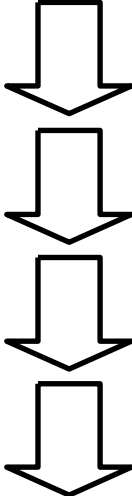
# INDEX EXAMPLE



Condition: `SELECT NumberOfKaons>700`

NTUPLE


Read all events



EventID	NumberOfKaons
1	731
2	800
3	345
4	543
5	567

INDEX

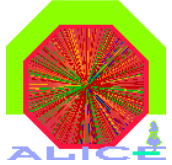
Read selected events



EventID	NumberOfKaons
3	345
4	543
5	567
1	731
2	800



## ALICE EVENT TAG SYSTEM

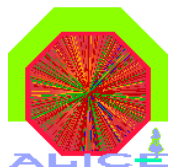


## ALICE TAG SYSTEM - PROTOTYPE

---



- Consists of 4 levels of information at the moment:
  - **RunTag fields:** Run specific information.
  - **LHCTag fields:** Information concerning the state of LHC per Alice run.
  - **DetectorTag fields :** Detector information per run.
  - **EventTag fields:** Information about each event.
- This is just a preliminary attempt.
- Any suggestions (I'm sure there will be many) are more than welcome!!!



## RunTag fields



### ● **AliceRunId**

- The run's unique identifier will be written in the raw data's header and will be retrieved from there (DAQ).

### ● **AliceMagneticField**

- The value of the magnetic field which will be retrieved from the DCS as values for different space-points (DCS)

### ● **AliceRunStartTime**

- This information will be stored in the header of the raw data (DAQ).

### ● **AliceRunStopTime**

- This information will be stored in the header of the raw data (DAQ).

### ● **AliceReconstructionVersion**

- Information from the reconstruction procedure (offline onformation).

### ● **AliceRunQuality**

- boolean, information coming from the validation script.

### ● **AliceBeamEnergy**

- Information about the cm beam energy that is read by the DCS directly from the machine and is stored in the db (DAQ or DCS)

### ● **AliceBeamType**: information about the colliding system type (pp, pA, AA)

- Will be stored in the statistics database or even in the header of the raw data or it will be retrived from the DCS db that reads it directly from the machine (DCS)

### ● **AliceCalibrationVersion**



## LHCTag fields

---



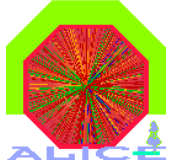
- **LHCState:**

- Maybe some comments describing the LHC run status ("test run" ...).

- **LHCLuminosity:**

- The luminosity value that will be read by the DCS directly from the LHC control.



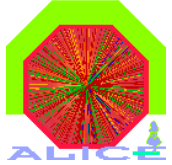


## DetectorTag fields

---



- Describes the detector configuration for each run.
- Consists of several fields that describe boolean parameters and show the status of each detector (active or not).
- Open questions concerning this level:
  - Should we have default configurations?
  - Should we have different sub-tables for each detector?
  - **Should we have a connection to the corresponding online detector db and retrieve the information from there?**

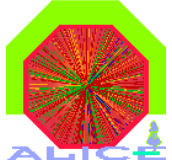


## EventTag fields (1)

---



- **AliceEventId**
- **GUID**: The file's unique identifier that comes directly from AliEn.
- **NumberOfParticipants**: Information coming from the ZDC.
- **ImpactParameter**: Calculated impact parameter from the number of participants.
- **PrimaryVertexX**
- **PrimaryVertexY**
- **PrimaryVertexZ**
- **TriggerInfo**
  
- **ZDCNeutronEnergy**: Reconstructed energy in the neutron ZDC.
- **ZDCProtonEnergy**: Reconstructed energy in the proton ZDC.
- **ZDCEMEnergy**: Reconstructed energy in the EM ZDC.
  
- **TOVertexZ**: Vertex Z position estimated by the START.



## EventTag fields (2)

---



- **NumberOfTracks:** Total multiplicity
- **NumberOfPositiveTracks**
- **NumberOfNegativeTracks**
- **NumberOfNeutralTracks**
- **NumberOfVOs**
- **NumberOfCascades**
- **NumberOfKinks**
- **NumberOfPMDTracks**
- **NumberOfPHOSTracks**
- **NumberOfEMCALTracks**
- **NumberOfFMDTracks**



## EventTag fields (3)

---



- **NumberOfJetCandidates**
- **NumberOfHardPhotonsCandidates**
- **NumberOfElectrons**
- **NumberOfMuons**
- **NumberOfPions**
- **NumberOfKaons**
- **NumberOfProtons**
- **NumberofLambdas**
- **KOPeakPosition**
- **KOPeakWidth**

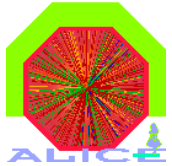


## EventTag fields (4)

---



- **NumberOfJPsiCandidates**
- **NumberOfPsiPrimeCandidates**
- **NumberOfUpsilonCandidates**
- **NumberOfUpsilonPrimeCandidates**
- **NumberOfCharmParticleCandidates**
- **NumberOfBeautyParticleCandidates**
- **TotalP**
- **MeanPt**
- **MaxPt**
- **FlowV1**
- **FlowV2**



## AliRunTag Class



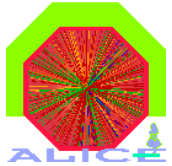
- **Constructor:** `AliRunTag();`
- **Set methods:**
  - `void SetRunId(Int_t Pid) {fAliceRunId = Pid;}`
  - `void SetMagneticField(Float_t Pmag) {fAliceMagneticField = Pmag;}`
  - `void SetBeamType(char *Ptype) {strcpy(fAliceBeamType,Ptype);}`
  - `void SetLHCTag(Float_t Plumin, char *type);`
- **Get methods:**
  - `Int_t GetRunId() {return fAliceRunId;}`
  - `Float_t GetMagneticField() {return fAliceMagneticField;}`
  - `char *GetBeamType() {return fAliceBeamType;}`
  - `LHCTag *GetLHCTag() { return &fLHCTag; }`
  - `EventTag *GetEventTag() { return &fEvTag; }`
- **An additional method that fills the EventTag class (TclonesArray in AliRunTag)**
  - `void AddEventTag(AliEventTag *t);`



## AliLHCTag Class



- **Constructor:** `AliLHCTag();`
- **Set methods:**
  - `void SetLHCState(char *type) {strcpy(fLHCState,type);}`
  - `void SetLuminosity(Float_t lumin) {fLHCLuminosity = lumin;}`
  - `void SetLHCTag(Float_t lumin, char *type) {fLHCLuminosity = lumin; strcpy(fLHCState,type); }`
  - `void SetLHCTag(Float_t Plumin, char *type);`
- **Get methods:**
  - `Float_t GetLuminosity() {return fLHCLuminosity;}`
  - `char *GetLHCState() {return fLHCState;}`

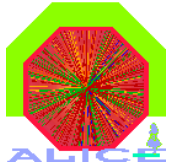


## AliDetectorTag Class



- **Constructor:**
  - `AliDetectorTag();`
  - `AliEventTag(AliEventTag *t);` :: Copy constructor
- **Set methods:**
  - `void SetITS(Int_t n) {fITS = n;}`
  - `void SetTPC(Int_t n) {fTPC = n;}`
- **Get methods:**
  - `Int_t GetITS() {return fITS;}`
  - `Int_t GetTPC() {return fTPC;}`
- **An additional private method that fills the `AliDetectorTag` clones.**
  - `virtual void CopyTag(AliDetectorTag *DetTag);`
    - ➔ `SetITS(DetTag->GetITS());`
    - ➔ `SetTPC(DetTag->GetTPC());`





## AliEventTag Class



- **Constructors:**

- `AliEventTag();` :: Initializes the `AliEventTag` fields
- `AliEventTag(AliEventTag *t);` :: Copy constructor

- **Set methods:**

- `void SetEventId(Int_t Pid) {fAliceEventId = Pid;}`
- `void SetGUID(Int_t Pid) {fGUID = Pid;}`
- `void SetNumOfParticipants(Int_t P) {fNumberOfParticipants = P;}`

- **Get methods:**

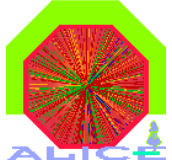
- `Int_t GetEventId() {return fAliceEventId;}`
- `Int_t GetGUID() {return fGUID;}`

- **An additional private method that fills the `AliEventTag` clones.**

- `virtual void CopyTag(AliEventTag *EvTag);`
  - ➔ `SetEventId(EvTag->GetEventId());`
  - ➔ `SetGUID(EvTag->GetGUID());`



## PROPOSED IMPLEMENTATION

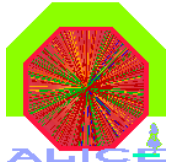


## PROPOSED SCENARIOS

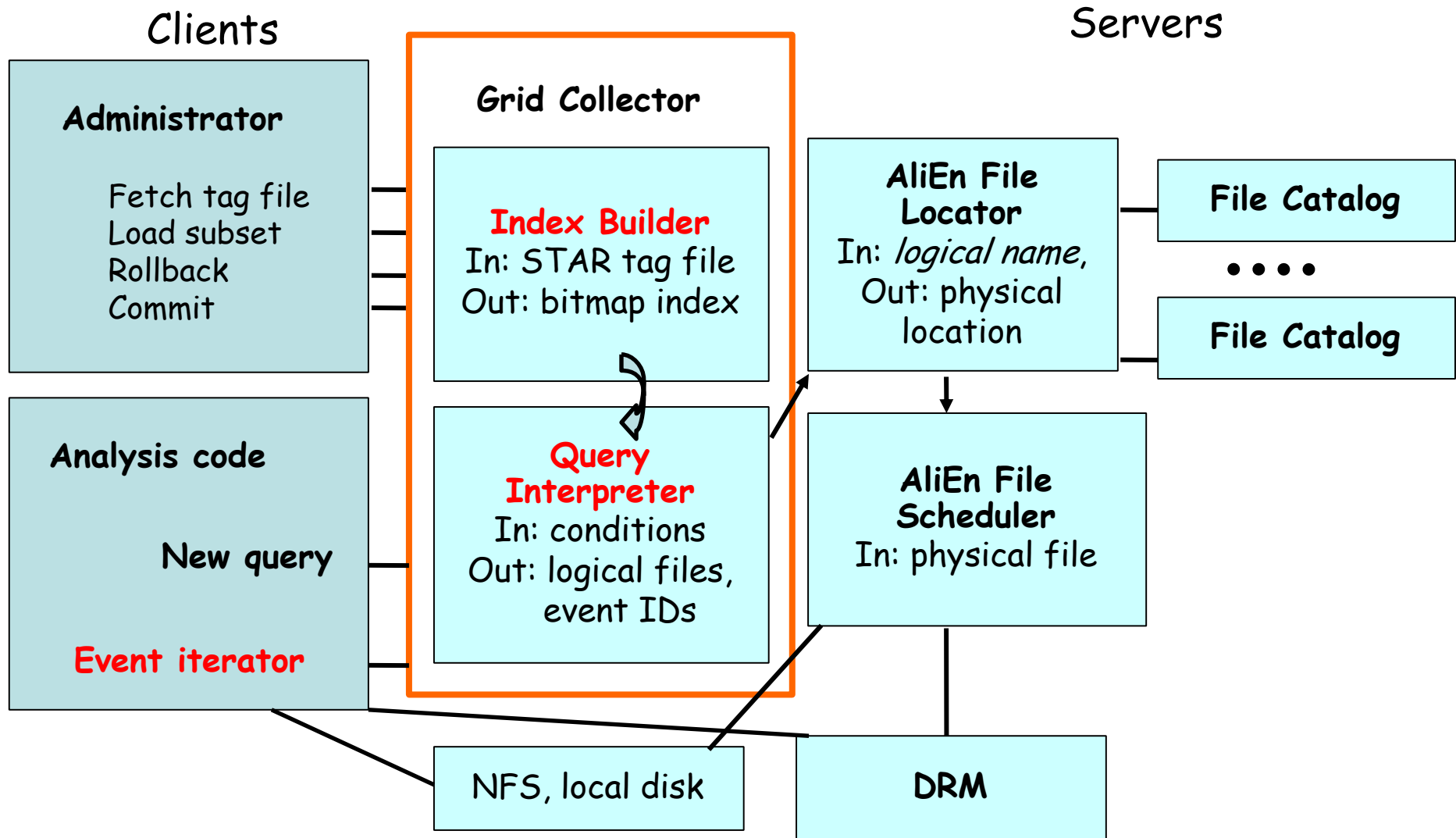
---

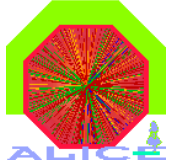


- "On the fly" creation:
  - Run the reconstruction code and create the ESDs.
  - Wait until the ESD is registered in AliEn and then perform an extra step in order to retrieve the corresponding location and the GUID.
  - Create the tag.root file by adding also the unique identifier.
  - Register the tag.root file in AliEn.
  
- Post creation:
  - Run the reconstruction code, create the ESD and register it in AliEn.
  - Run a fast and efficient "post process" that will loop over all ESDs of a run and fill the tag.root files.
  - Register the tag.root file in AliEn.
  
- Use the GC in order to produce the corresponding indices for every attribute that will be stored in the tag.root file.
  
- Use the GC in the analysis framework in order to have fast and efficient access to the events of interest by imposing selection criteria inside the analysis macro.



# PROPOSED ARCHITECTURE





## SUMMARY

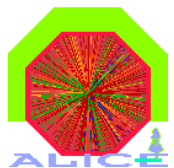
---

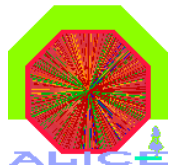


- STAR collaborators use *GC* in order to select and analyze specific events and not the whole data sample.
- *GC*'s component that builds the indices will be included in ROOT's new version.
- ALICE's event tag prototype consists of ~70 parameters but more will be added.
- Two proposed scenarios for the implementation in ALICE:
  - "On the fly" production of tag.root files.
  - Post production of tag.root files.

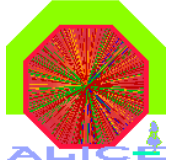
## NEXT STEPS

- Feedback from the PWG concerning parameters that they might want to include.





## BITMAP INDEX



## BITMAP INDEX

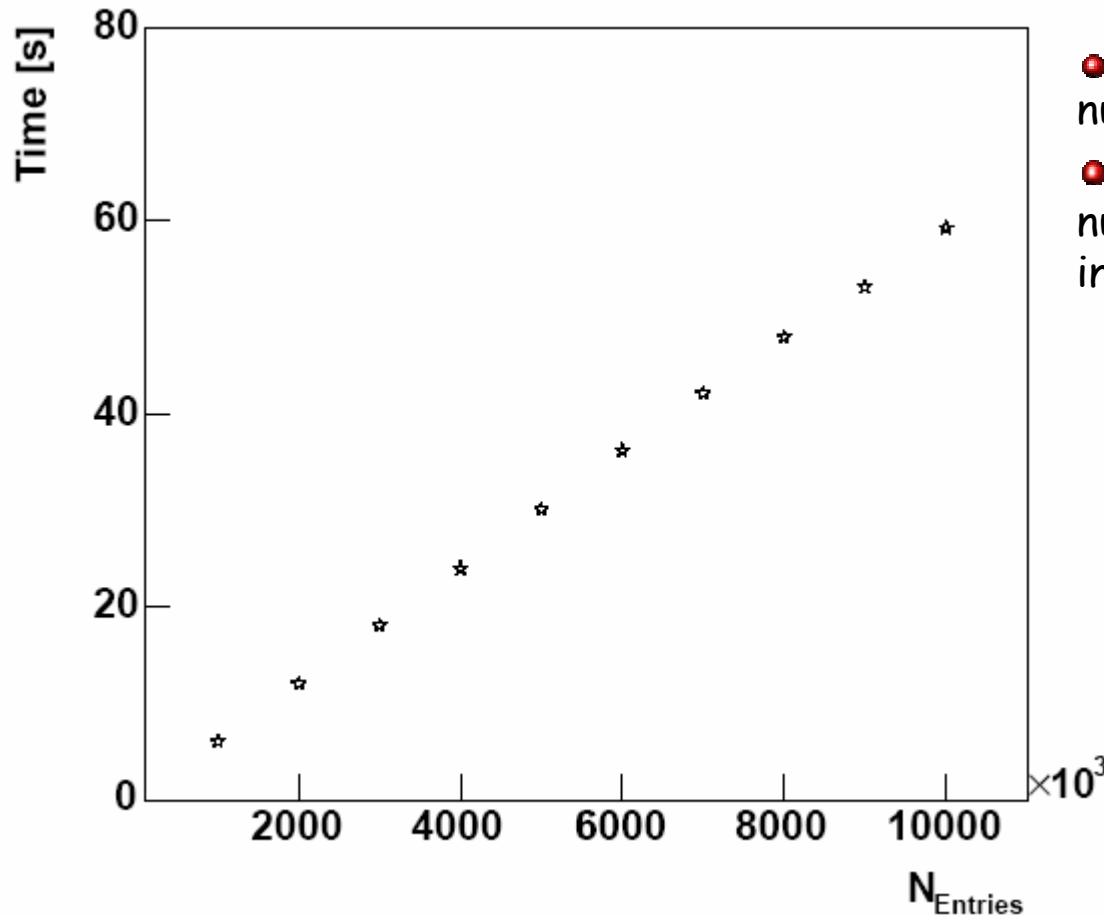


- Kurt Stockinger @LBNL is working on this subject.
- It is going to be implemented in ROOT's new version that will be released soon.
- Thanks to Kurt who sent the code I've managed to run a few examples and tests:
  - Create a root file that consists of a tree that has two attributes (one integer & one float).
  - Increased the number of entries and calculated the time needed in order to build the indices.
  - Increased the number of entries and compared the time needed to evaluate a query when using:
    - TTree::Draw
    - TBitmapIndex::Draw
  - Check the size of the index files with respect to the number of entries .





# BITMAP INDEX TESTS - BUILDING TIME



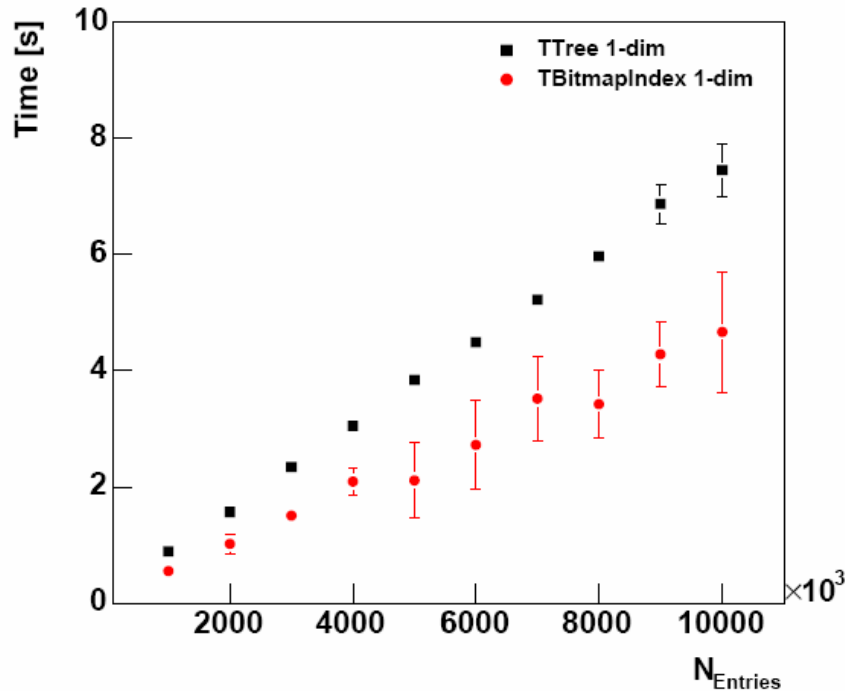
- Building time is proportional to the number of entries.
- This time depends also on the number of parameters that are used in order to build the indices.



# BITMAP INDEX TESTS - TIME COMPARISON

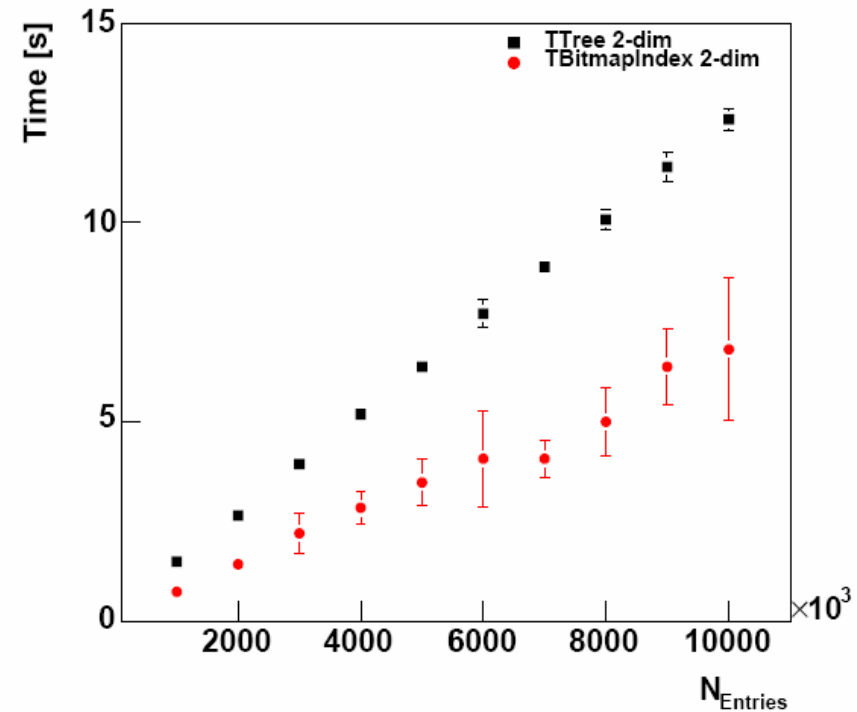


## One dimensional query



- Both calculated times are proportional to the number of entries.
- Bitmap index evaluates the 1-dim query 40% faster than TTree.

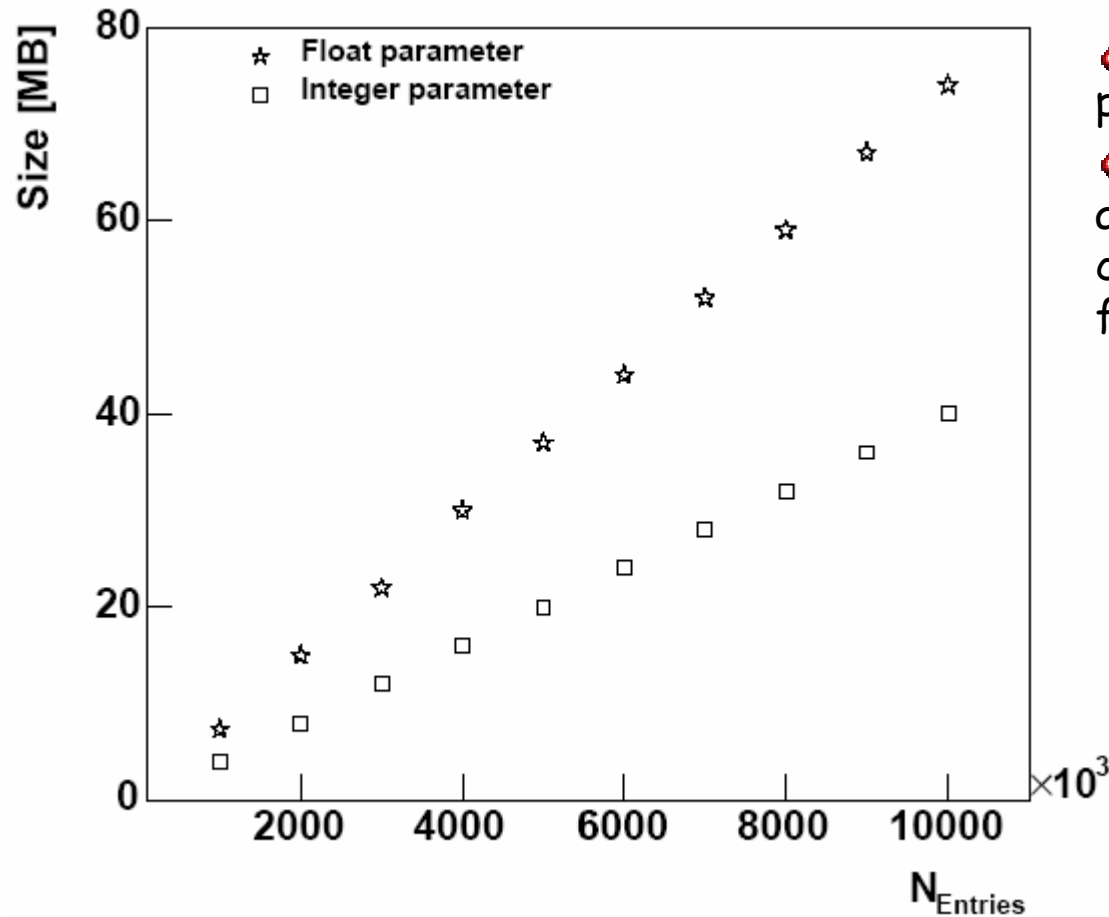
## Multi dimensional query



- Both calculated times are proportional to the number of entries.
- Bitmap index evaluates the 2-dim query 49% faster than TTree.



# BITMAP INDEX TESTS - SIZE



- Both calculated sizes are proportional to the number of entries.
- The size of the integer index file is almost 50% smaller than the corresponding size of the float index file.